Teoría de TADs

Pablo Castro
Estructura de Datos y Alg.
UNRC

Tipos y Algebras

Un algebra es un conjunto soporte más operaciones:

Conjunto soporte

$$\langle \mathbb{N}, +, 0 \rangle$$

Operaciones

Las operaciones son funciones:

Las constantes se pueden ver como operaciones sin parámetros

$$+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$0 : \to \mathbb{N}$$

Pueden ser operaciones parciales

Tipos y Algebras

Los tipos en un lenguaje de programación son algebras:

$$\langle \mathbb{Z}, +, -, *, 0, 1 \rangle$$

Enteros

$$\langle Bool, \vee, \wedge, \neg, true, false \rangle$$

Booleanos

$$\langle [A], :, [] \rangle$$

Listas polimorficas de tipo A

$$\langle \{A\}, \cup, \cap, \setminus, \emptyset \rangle$$

Conjuntos polimorficos de tipo A

Algebras Heterogéneas

Muchas veces utilizaremos algebras que tienen varios conjuntos soportes

$$\langle \{A\}, Bool, \cup, \cap, \in, \emptyset \rangle$$

Conjunto de interes

Conjunto auxiliar

$$\in : \{A\} \times A \to Bool$$

Retorna un elemento booleano que no pertenece al conjunto soporte

Clases de Operaciones

Observadoras: Aquellas operaciones que devuelven un elemento de un tipo auxiliar

$$\in : \{A\} \times A \to Bool$$

Modificadoras: Aquellas operaciones que toman al menos un elemento del tipo de interés, y devuelven algo del tipo de interés

$$\cup : \{A\} \times \{A\} \to \{A\}$$

Generadoras: Aquellas que no toman un elemento del tipo de interés, y devuelven algo del tipo de interés

$$\emptyset : \to \{A\}$$

Descripción de TADs

Un TAD es una descripción abstracta de un tipo. Se especifican con axiomas:

Interface de la pila

$$
\begin{aligned}
&new :\to Stack\\
&push : Stack \times El \to Stack\\
&pop : Stack \to Stack\\
&top : Stack \to El\\
&empty : Stack \to Bool
\end{aligned}
$$

$$
\begin{aligned}
&pop(new()) = new()\\
&pop(push(s, e)) = s\\
&top(new()) = error\\
&top(push(s, e)) = e\\
&empty(new()) = true\\
&empty(push(s, e)) = false
\end{aligned}
$$

Axiomas que definen sus propiedades

Algebra de Términos

Podemos construir un álgebra utilizando los términos provistos por la interface:

$$\boxed{\begin{array}{c} new() \\ pop(push(new(), 1)) \end{array}}$$

$$push((push(new(), 1), 2)$$

$$push((push(new(), 0), 0)$$

$$\boxed{\begin{array}{c} top(push(new(), 0)) \\ 0 \end{array}}$$

$$push(new(), 1)$$

$$push(new(), 0)$$

Tenemos un número infinito de términos

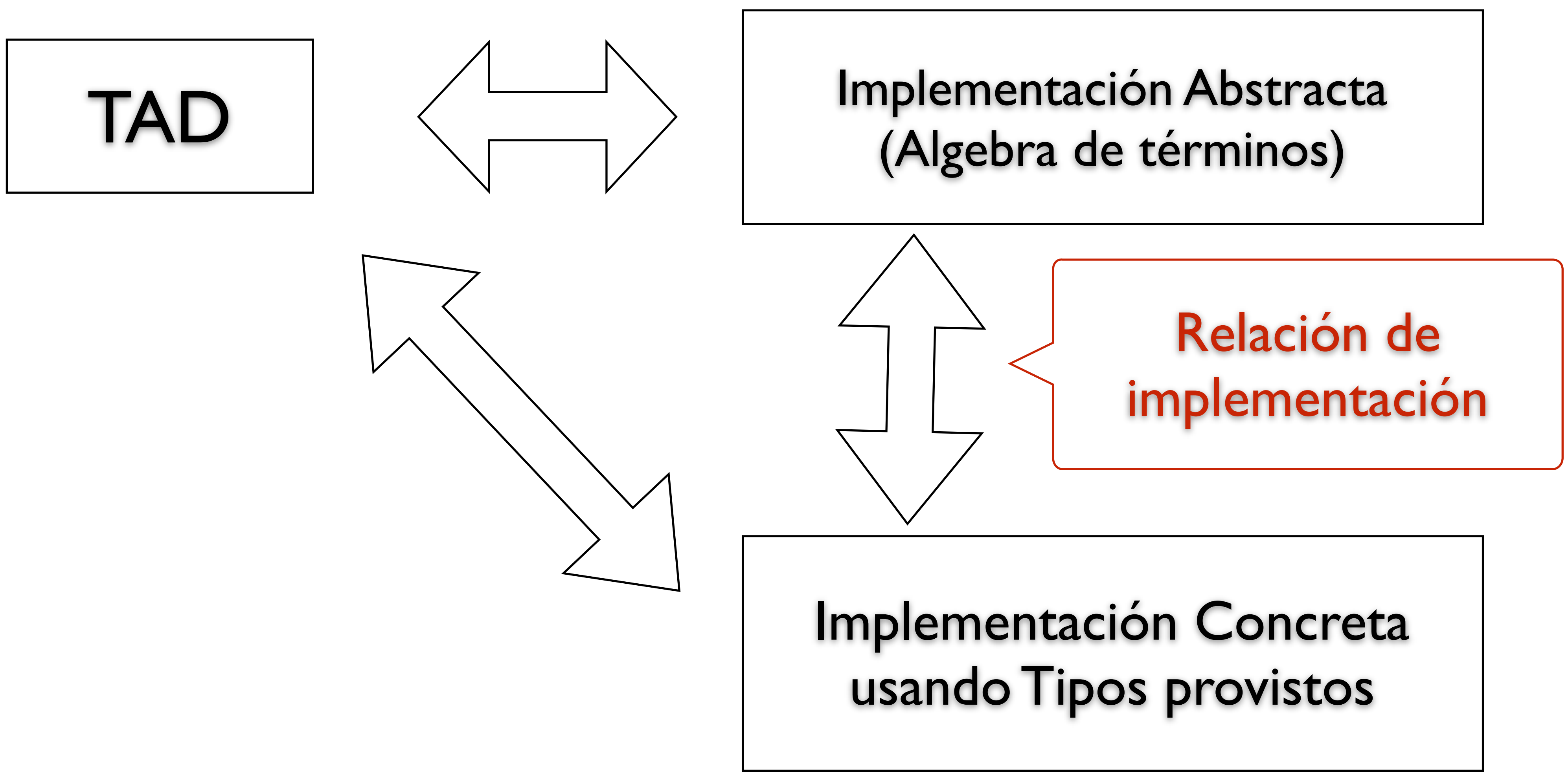
Podemos agrupar aquellos términos que son equivalentes por los axiomas

El álgebra de términos denotada: $T(X)/\equiv$

y satisface los axiomas

Implementación de TADS

Muchas veces los tipos que necesitamos no están provistos en los lenguajes de programación.



Implementación de TADS

- Llamaremos al tipo que queremos implementar <u>Algebra Abstracta</u>
- Llamaremos a su implementación <u>Algebra Concreta</u>

La relación entre ambas algebras viene dada por una función de abstracción:

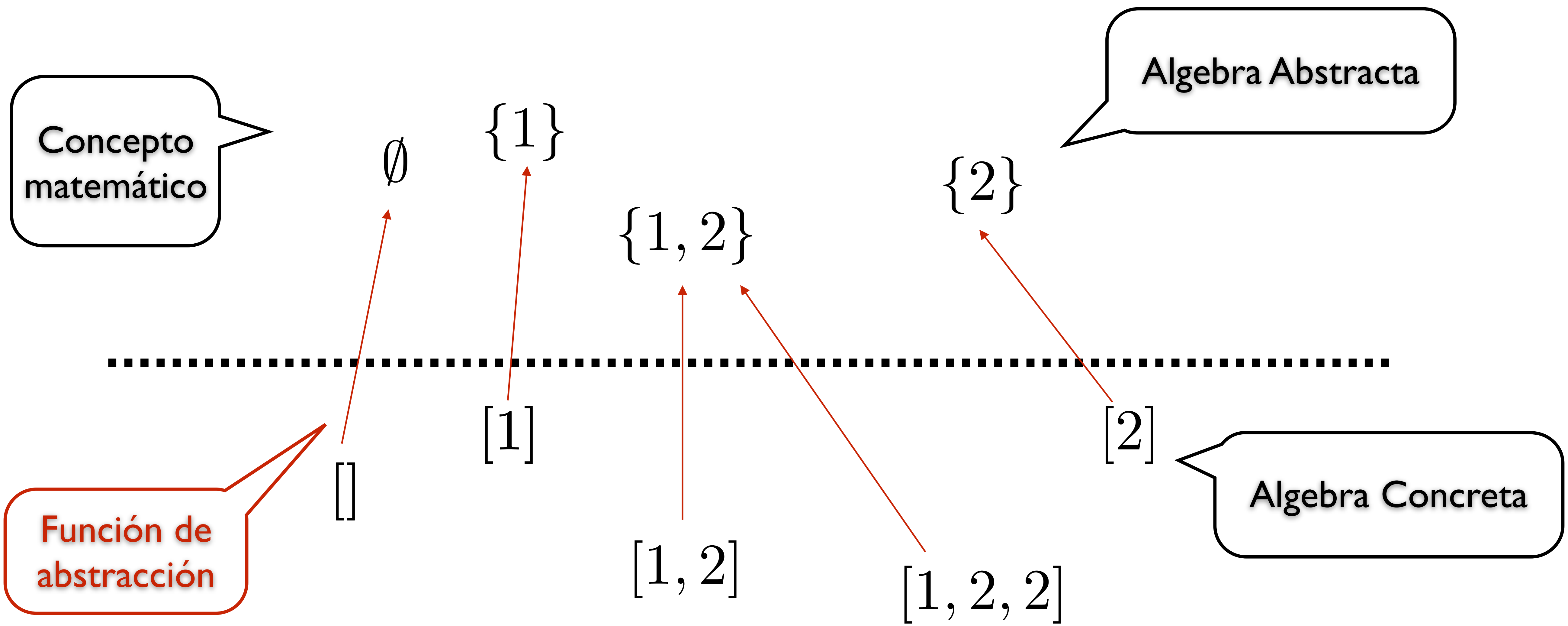
$$[\![-]\!] : C \to A$$

Algebra concreta

Algebra abstracta

Ejemplo

El tipo de conjuntos en general no viene provisto por el lenguaje de programación.



Se implementan conjuntos con listas.

Función de Abstracción

Relaciona un tipo con su implementación:

$$[\![-]\!] : C \to A$$

Debe cumplir:

- Es suryectiva.
- Cada operación del algebra abstracta tiene una correspondiente en el algebra concreta

(y cumplir ciertos requisitos…)

Operaciones Generadoras

Para cada operación generadora del algebra abstracta:

$$f : T \to A$$

Debe haber una operación generadora del algebra concreta, tal que:

$$\llbracket f_i'(c) \rrbracket = f_i(c)$$

En diagramas:

$$\begin{tikzcd}
T \arrow[r, "f"] \arrow[dr, "f'"'] & A \\
& C \arrow[u, "\llbracket\rrbracket"']
\end{tikzcd}$$

Mundo Abstracto

Mundo Concreto

Operaciones Modificadoras

Para cada operación modificadora abstracta:

$$f : T \times A \to A$$

Tiene que haber una modificadora concreta:

$$f' : T \times C \to C$$

Tal que: $[\![f'(x, c)]\!] = f(x, [\![c]\!])$

O en diagramas:

$$\begin{tikzcd}
T \times A \arrow[r, "f"] & A \\
T \times C \arrow[u, "id \times [\![\,]\!]"] \arrow[r, "f'"'] & C \arrow[u, "[\![\,]\!]"']
\end{tikzcd}$$

Mundo Abstracto

Mundo Concreto

Operaciones Observadoras

Para cada operación observadora abstracta:

$$f : S \times A \to T$$

Debe haber una operación observadora concreta:

$$f' : S \times C \to T$$

Tal que: $f'(x, c) = f(x, [\![c]\!])$

En diagramas:

$$\begin{tikzcd} S \times A \arrow[r, "f"] & T \\ S \times C \arrow[u, "id \times [\![]\!]"] \arrow[ur, "f'"'] & \end{tikzcd}$$

Ejemplo

Supongamos el algebra de los conjuntos:

$$\langle \{A\}, \cup, \cap, \in, \emptyset \rangle$$

Podemos implementarla con listas:

Conjuntos representados por listas

$$\langle [A], \overline{\cup}, \overline{\cap}, \overline{\in}, \overline{\emptyset} \rangle$$

Implementación de las operaciones

Definida recursivamente

Función de abstracción:

$$[\![[\,]]\!] = \emptyset$$

$$[\![x : xs]\!] = \{x\} \cup [\![xs]\!]$$

Ejemplo (cont.)

Debemos dar la implementación de cada operación, por ejemplo:

$$[]\overline{\cup}ys = ys$$

$$(x : xs)\overline{\cup}ys = x : (xs\overline{\cup}ys)$$

Repite elementos!

Deberíamos demostrar:

$$[\![xs\overline{\cup}ys]\!] = [\![xs]\!] \cup [\![ys]\!]$$

$$[\![xs\overline{\cap}ys]\!] = [\![xs]\!] \cap [\![ys]\!]$$

Es decir, las operaciones conmutan

$$x\overline{\in}xs = x \in [\![xs]\!]$$

Otro Ejemplo

Consideremos el algebra de los enteros

$$\langle Int, 0, +, -(unaria)\rangle$$

Podemos implementarla con los naturales:

$$\langle (Nat, Nat), \overline{0}, \oplus, \ominus, \rangle$$

La función de abstracción es:

$$[\![(m, n)]\!] = m - n$$

Ejemplo (cont.)

Las operaciones podemos definirlas como:

$$(m, n) \oplus (m', n') = (m + m', n + n')$$

$$\ominus(n, m) = (m, n)$$

$$\overline{0} = (0, 0)$$

Demostremos la corrección de la suma:

$$\begin{array}{c}
[\![(n, m) \oplus (p, q)]\!] \\
= [\text{def.}\oplus] \\
[\![(n + p, m + q)]\!] \\
= [\text{def.}[\![]\!]] \\
n + p - (m + q) \\
= [\text{Arit.}] \\
n - m + p - q \\
= [\text{def.}[\![]\!]] \\
[\![(n, m)]\!] + [\![(p, q)]\!]
\end{array}$$

Ejemplo III

Supongamos que queremos implementar los booleanos:

$$\langle Bool, true, false, \wedge, \vee, \neg \rangle$$

Los implementaremos con los naturales:

$$\langle Nat, \overline{true}, \overline{false}, \overline{\wedge}, \overline{\vee}, \overline{\neg} \rangle$$

En donde:

$$[\![n]\!] = (n \neq 0)$$

Cualquier numero distinto de 0 representa true, y 0 representa false

Ejemplo (III)

Definamos las operaciones:

$$\overline{true} = 8$$

$$\overline{false} = 0$$

$$p \overline{\wedge} q = p * q$$

$$p \overline{\vee} q = p + q$$

$$\overline{\neg} p = if \; p = 0 \to 1$$
$$\qquad\qquad \square \; p \neq 0 \to 0$$

Ejercicio: Demostrar su corrección.

Propiedades Formales

Dada una función de abstracción:

$$[\![ ]\!] : C \to A$$

Suryectiva

Hay una relación de equivalencia asociada:

$$x \equiv_{[\![ ]\!]} y \equiv [\![x]\!] = [\![y]\!]$$

Y una algebra modulo esta equivalencia:

$$C \backslash \equiv_{[\![ ]\!]}$$

Que se comporta igual que el algebra abstracta.

Invariantes de Representación

Consideremos el siguiente ejemplo:

$$
\begin{tikzcd}
\langle \mathbb{Q}, +, -, *, \backslash, =, 0 \rangle \\
\langle \mathbb{N} \times \mathbb{N}, \overline{+}, \overline{-}, \overline{*}, \overline{\backslash}, \overline{=}, \overline{0} \rangle \arrow[u, "\text{Función de abstracción}"']
\end{tikzcd}
$$

$$(1,2) \to \frac{1}{2} \qquad (2,4) \to \frac{2}{4} \qquad (8,1) \to \frac{8}{1} \qquad (0,1) \to \frac{0}{1}$$

No representa nada $(2, 0)$

Invariantes de Representación

Un invariante de representación es un predicado sobre el algebra concreta que permite caracterizar aquellos elementos que representan elementos del algebra abstracta

Es decir, es una función:

$$inv : C \to Bool$$

Invariantes de Representación

Los invariantes de representación deben cumplir ciertos requisitos:

- Para cada operación generadora: $g : T \to C$

$$\forall t \in T : inv(g(t))$$

se generan elementos validos

- Para cada modificadora: $m : T \times C \to C$

$$\forall c \in C, t \in T : inv(c) \Rightarrow inv(m(t, c))$$

se retornan elementos validos

- Además:

$$\forall a \in A : \exists c \in C : inv(c) \wedge [\![c]\!] = a$$

Ejemplo

Retomando el ejemplo de los racionales:

$$[\![(n, d)]\!] = \frac{n}{d}$$

El invariante es:

$$inv(n, d) = d \neq 0$$

Solo los pares cuyo segundo elemento es diferente a 0, representan racionales

Ejercicio: definir las operaciones.

Implementación

En Java tenemos la siguiente correspondencia:

Algebra Abstracta → Interface

Función de abstracción

Implementación

Algebra Concreta → Clase

El invariante de representación es un método repOk, diciendo que objetos son correctos

La función de abstracción puede ser un función toString() que describa que valor representa cada objeto

Un Ejemplo

Consideremos los racionales:

```
/**
*   Una interface para racionales, para ilustrar las nocion de funcion de asbtraccion
*   @author Pablo Castro
*/

public interface Racional{

    /**
    *   Operacion para suma racionales
    */
    public void suma(Racional r);

    /**
    *   Operacion para multiplicacion de racionales
    */
    public void mult(Racional r);

    /**
    *   operacion para resta de racionales
    */
    public void neg();

    /**
    *   Operacion para dividir racionales
    */
    public void div(Racional r);

}
```

Podemos agregar todas las operaciones que queramos

Una Implementación

```java
public class RacionalPar implements Racional{

    private int num; // el numerador del racional
    private int den; // el denominador

    /**
    * Un constructor basico para racionales
    * @pre d != 0
    */
    public RacionalPar(int n, int d){
        this.num = n;
        this.den = d;
    }

    /**
    * Observadora, retorna el numerador
    */
    public int getNum(){
        return num;
    }

    /**
    * Observadora retorna el denominador
    */
    public int getDen(){
        return den;
    }
```

Implementamos racionales con pares

Métodos observadores

Una Implementación (I)

```java
/**
 * Suma un racional al actual
 * @param    r   el racional para sumar
 * @pre r debe ser RacionalPar y r.num !=0
 * @return la suma
 */
public void suma(Racional r){
    if (r instanceof RacionalPar){
        RacionalPar param = (RacionalPar) r;
        this.num = (this.num * param.getDen()) + (param.getNum() * this.den);
        this.den = this.den * param.getDen();
    }
    else
        throw new RacionalExcepcion();
}
```

Implementamos cada una de las operaciones

El chequeo de tipos en tiempo de ejecución se puede evitar usando genericidad

Hay que tener cuidado con los tipos que se devuelven y de los parámetros!

Una Implementación (II)

toString es una forma de describir la función de abstracción

```java
    /**
     * La funcion de abstraccion
     * @return   el numero que representa el objeto actual
     */
     public String toString(){
         return String.valueOf((float) this.num / (float) this.den);
     }

     /**
     * El invariante de representacion
     * @return true cuando el segundo componente del par es diferente a 0
     */
     public boolean repOk(){
         return (den != 0);
     }

}// end of class
```

repOk es el invariante de clase

Implementación con Genericidad

```java
/**
*   Una interface simple de los racionales usando
*   genericidad
*/

public interface Racional<T extends Racional>{

     /**
    *   Operacion para suma racionales
    */
    public void suma(T r);

    /**
    *   Operacion para multiplicacion de racionales
    */
    public void mult(T r);

    /**
    *   operacion para resta de racionales
    */
    public void neg();

    /**
    *   Operacion para dividir racionales
    */
    public void div(T r);


}
```

Usamos genericidad para evitar hacer castings

Implementación con Genericidad II

```java
/**
*  Una implementacion de los racionales con genericidad
*
*/
public class RacionalPar implements Racional<RacionalPar>{
    private int num; // el numerador
    private int den; // el denominador

    // Implementar el resto de los metodos
   /**
    * Multiplica un racional al actual
    * @param    r   el racional para multiplica
    * @pre r debe ser RacionalPar y r.num !=0
    * @return la multiplicacion
    */
    public void mult(RacionalPar r){
            this.num = this.num * r.getNum();
            this.den = this.den * r.getDen();
    }
    /**
    * La funcion de abstraccion
    * @return   el numero que representa el objeto actual
    */
    public String toString(){
        return String.valueOf((float) this.num / (float) this.den);
    }
    /**
    * El invariante de represantacion
    * @return true cuando el segundo componente del par es diferente a 0
    */
    public boolean repOk(){
        return (den != 0);
    }
}
```

No necesitamos hacer castings